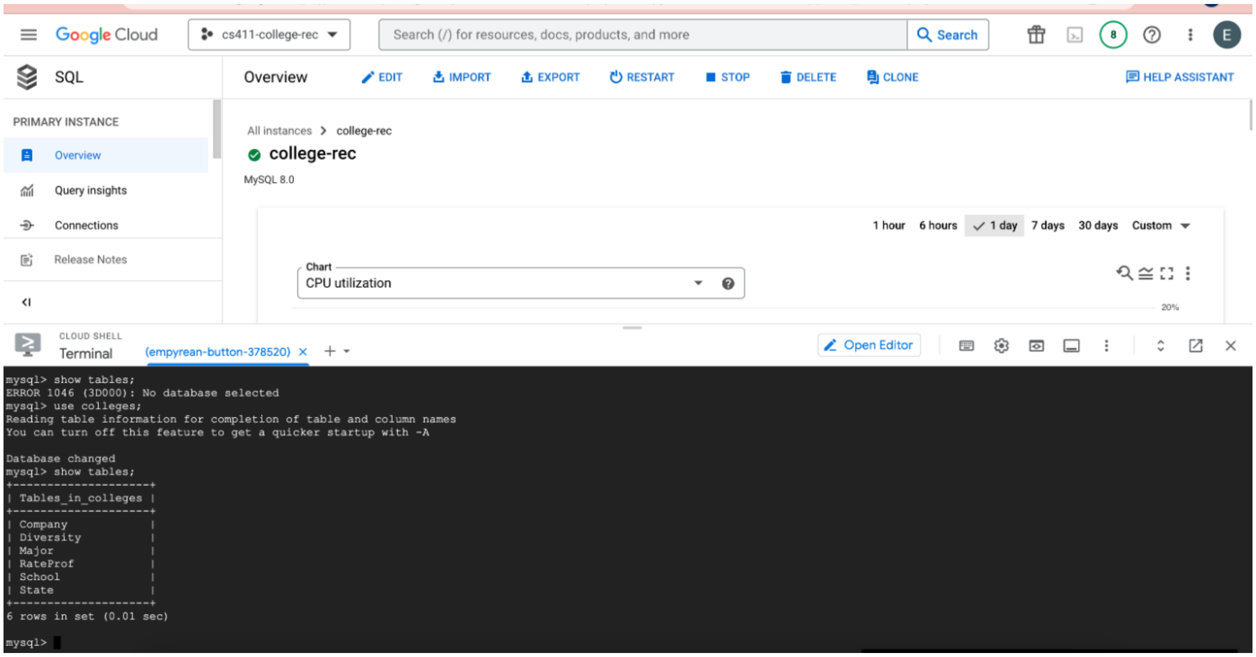
```
CREATE TABLE School (
        Name                    VARCHAR(256) NOT NULL,
        TotalEnrollment         INTEGER,
        State                   VARCHAR(20),
        StateSalaryRank         INTEGER,
        EarlyCareerPay          REAL,
        MidCareerPay            REAL,
        STEMPercent             INTEGER,
        Type                    VARCHAR(20),
        DegreeLength            VARCHAR(10),
        BoardCost               REAL,
        InStateTuition          REAL,
        OutStateTuition         REAL
        PRIMARY KEY(Name)
);

CREATE TABLE Diversity (
        Name                    VARCHAR(256) NOT NULL,
        WomenCount              INTEGER,
        AIANCount               INTEGER,
        AsianCount              INTEGER,
        BlackCount              INTEGER,
        HispanicCount           INTEGER,
        PacificCount            INTEGER,
        WhiteCount              INTEGER,
        PRIMARY KEY(Name),
        FOREIGN KEY(Name) REFERENCES School(Name) ON DELETE CASCADE
);

CREATE TABLE State (
        StateName               VARCHAR(20) NOT NULL,
        Population              INTEGER,
        CrimeRate               REAL,
        HappinessScore          REAL,
        Code                    CHAR(2),
        PRIMARY KEY(StateName)
);

CREATE TABLE Company (
        CompanyName             VARCHAR(256) NOT NULL,
        Industry                VARCHAR(100),
        Location                VARCHAR(256),
        EmployeeCount           INTEGER,
        PRIMARY KEY(CompanyName)
);

CREATE TABLE RateProf (
        StudentID               INTEGER NOT NULL,
        ProfessorName           VARCHAR(50),
        SchoolName              VARCHAR(256),
        State                   CHAR(2),
        StarRating              REAL,
        DifficultyRating        REAL,
        CourseID                VARCHAR(20),
        DepartmentName          VARCHAR(256),
        PRIMARY KEY(StudentID)
);

CREATE TABLE Major (
        Major                   VARCHAR(256) NOT NULL,
        Ranking                 INTEGER,
        TotalCount              INTEGER,
        WomenCount              INTEGER,
        Category                VARCHAR(256),
        EmploymentCount         INTEGER,
        MedianSalary            REAL,
        PRIMARY KEY(Major)
);
```

Code version of DDL Commands:

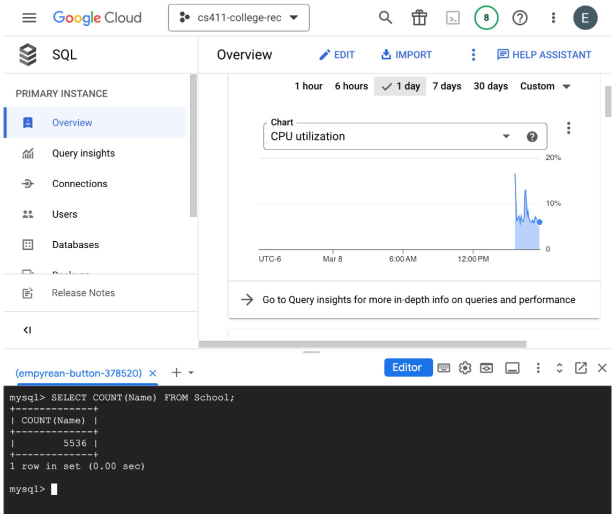
```
CREATE TABLE School (Name VARCHAR(256) NOT NULL, TotalEnrollment INTEGER,
State VARCHAR(20), StateSalaryRank INTEGER, EarlyCareerPay REAL,
MidCareerPay REAL, STEMPercent INTEGER, Type VARCHAR(20), DegreeLength
VARCHAR(10), BoardCost REAL, InStateTuition REAL, OutStateTuition REAL,
PRIMARY KEY(Name));

CREATE TABLE Diversity (Name VARCHAR(256) NOT NULL, WomenCount INTEGER,
AIANCount INTEGER, AsianCount INTEGER, BlackCount INTEGER, HispanicCount
INTEGER, PacificCount INTEGER, WhiteCount INTEGER, PRIMARY KEY(Name),
FOREIGN KEY(Name) REFERENCES School(Name) ON DELETE CASCADE);

CREATE TABLE State (StateName VARCHAR(20) NOT NULL, Population INTEGER,
CrimeRate REAL,  HappinessScore REAL, Code CHAR(2),PRIMARY KEY(StateName));

CREATE TABLE Company (CompanyName VARCHAR(256) NOT NULL, Industry
VARCHAR(100), Location VARCHAR(256),  EmployeeCount INTEGER, PRIMARY
KEY(CompanyName));

CREATE TABLE RateProf (ProfessorName VARCHAR(50), SchoolName VARCHAR(256),
State CHAR(2), StarRating REAL, DifficultyRating REAL, CourseID
VARCHAR(20), DepartmentName VARCHAR(256), StudentID INTEGER NOT NULL,
PRIMARY KEY(StudentID));

CREATE TABLE Major (Ranking INTEGER, Major VARCHAR(256) NOT NULL,
TotalCount INTEGER, WomenCount INTEGER, Category VARCHAR(256),
EmploymentCount INTEGER, MedianSalary REAL, PRIMARY KEY(Major));
```

Implemented Databases on GCP:

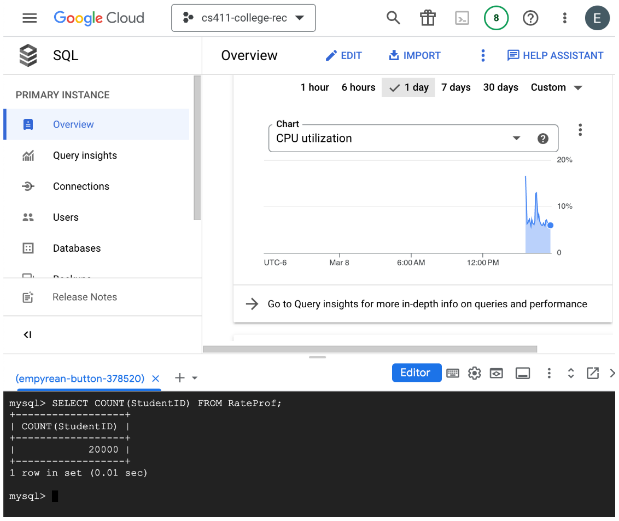


```
mysql> show tables;
ERROR 1046 (3D000): No database selected
mysql> use colleges;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+--------------------+
| Tables_in_colleges |
+--------------------+
| Company            |
| Diversity          |
| Major              |
| RateProf           |
| School             |
| State              |
+--------------------+
6 rows in set (0.01 sec)

mysql> 
```

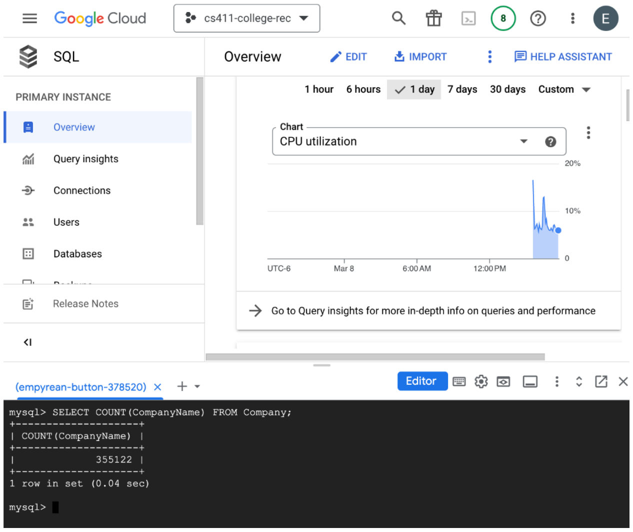
1000 rows of data in 3 tables



```
mysql> SELECT COUNT(Name) FROM School;
+-------------+
| COUNT(Name) |
+-------------+
|        5536 |
+-------------+
1 row in set (0.00 sec)

mysql> 
```



```
mysql> SELECT COUNT(StudentID) FROM RateProf;
+------------------+
| COUNT(StudentID) |
+------------------+
|            20000 |
+------------------+
1 row in set (0.01 sec)

mysql> 
```



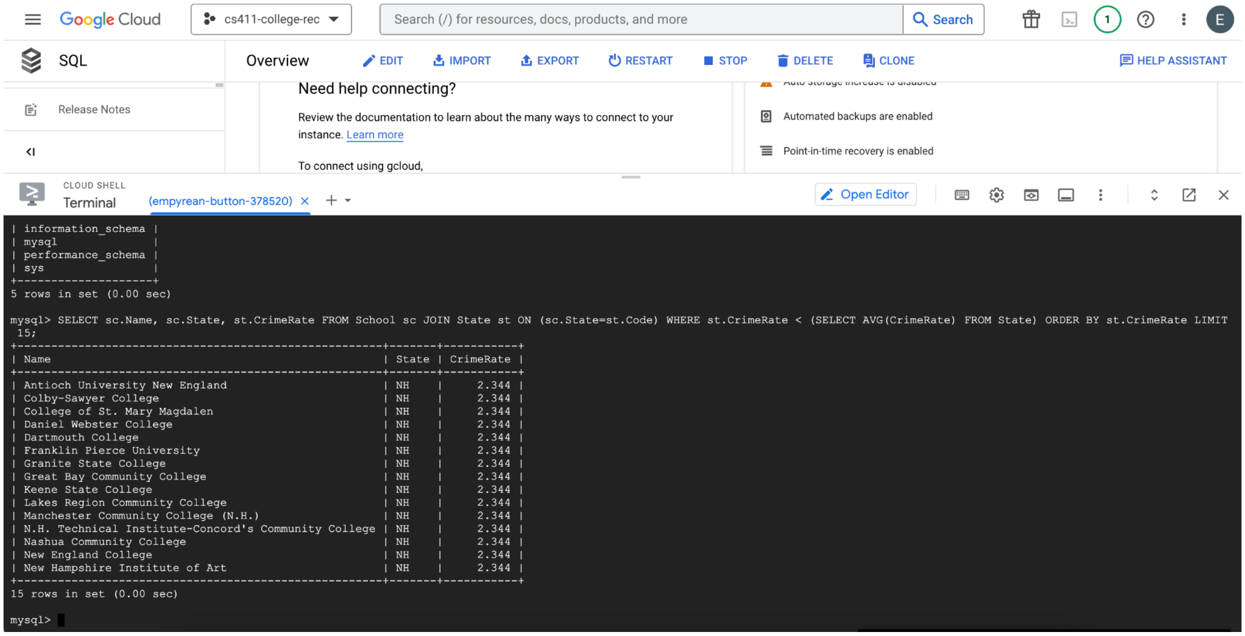
```
mysql> SELECT COUNT(CompanyName) FROM Company;
+--------------------+
| COUNT(CompanyName) |
+--------------------+
|             355122 |
+--------------------+
1 row in set (0.04 sec)

mysql> 
```

Advanced SQL Query 1

This is a SQL query that finds the schools in states that have a **lower crime rate** than the average crime rate across the US. Order by CrimeRate in ascending order.

SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;

- Join
- Subqueries



Need help connecting?

Review the documentation to learn about the many ways to connect to your instance. Learn more

To connect using gcloud,

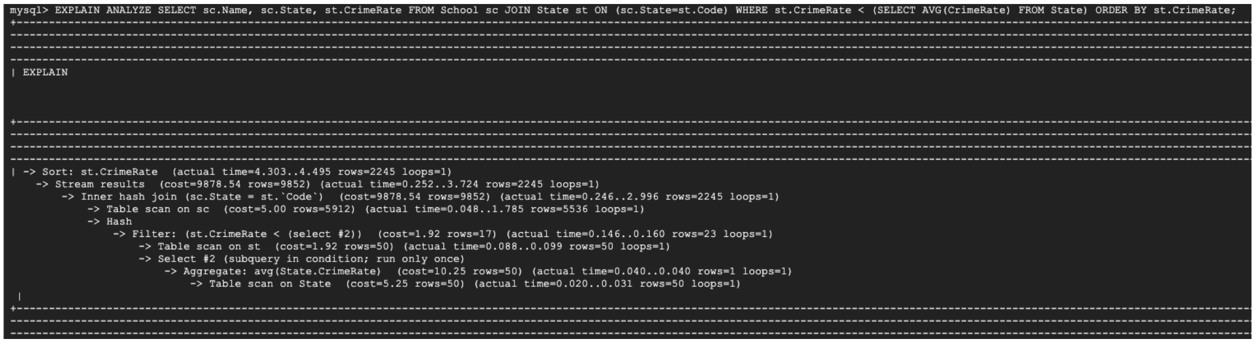
Auto storage increase is disabled

Automated backups are enabled

Point-in-time recovery is enabled

```
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate LIMIT
 15;
+------------------------------------------------------+-------+-----------+
| Name                                                 | State | CrimeRate |
+------------------------------------------------------+-------+-----------+
| Antioch University New England                       | NH    |     2.344 |
| Colby-Sawyer College                                 | NH    |     2.344 |
| College of St. Mary Magdalen                         | NH    |     2.344 |
| Daniel Webster College                               | NH    |     2.344 |
| Dartmouth College                                    | NH    |     2.344 |
| Franklin Pierce University                           | NH    |     2.344 |
| Granite State College                                | NH    |     2.344 |
| Great Bay Community College                          | NH    |     2.344 |
| Keene State College                                  | NH    |     2.344 |
| Lakes Region Community College                       | NH    |     2.344 |
| Manchester Community College (N.H.)                  | NH    |     2.344 |
| N.H. Technical Institute-Concord's Community College | NH    |     2.344 |
| Nashua Community College                             | NH    |     2.344 |
| New England College                                  | NH    |     2.344 |
| New Hampshire Institute of Art                       | NH    |     2.344 |
+------------------------------------------------------+-------+-----------+
15 rows in set (0.00 sec)

mysql> 
```

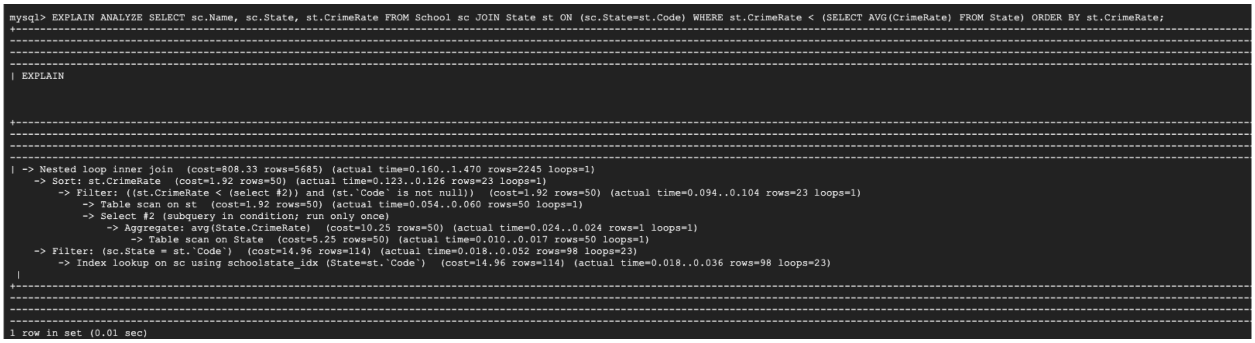
Before indexing, the cost is **9878.54**



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Sort: st.CrimeRate  (actual time=4.303..4.495 rows=2245 loops=1)
    -> Stream results  (cost=9878.54 rows=9852) (actual time=0.252..3.724 rows=2245 loops=1)
        -> Inner hash join (sc.State = st.`Code`)  (cost=9878.54 rows=9852) (actual time=0.246..2.996 rows=2245 loops=1)
            -> Table scan on sc  (cost=5.00 rows=5912) (actual time=0.048..1.785 rows=5536 loops=1)
            -> Hash
                -> Filter: (st.CrimeRate < (select #2))  (cost=1.92 rows=17) (actual time=0.146..0.160 rows=23 loops=1)
                    -> Table scan on st  (cost=1.92 rows=50) (actual time=0.088..0.099 rows=50 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(State.CrimeRate)  (cost=10.25 rows=50) (actual time=0.040..0.040 rows=1 loops=1)
                            -> Table scan on State  (cost=5.25 rows=50) (actual time=0.020..0.031 rows=50 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Design 1:
CREATE INDEX schoolstate_idx on School(State);

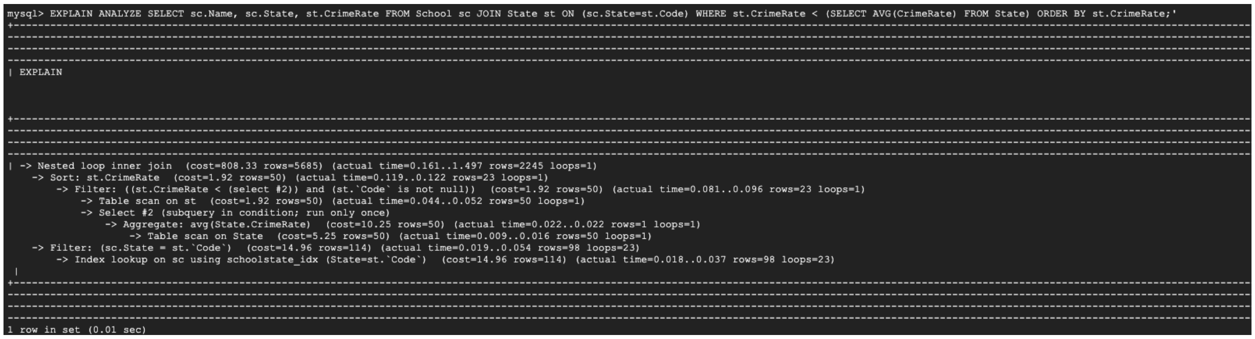
The cost decreased from 9878.54 to **808.33** which is a big improvement! And it only took 1.47 seconds to access all the rows. This index seems important to keep.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=808.33 rows=5685) (actual time=0.160..1.470 rows=2245 loops=1)
    -> Sort: st.CrimeRate  (cost=1.92 rows=50) (actual time=0.123..0.126 rows=23 loops=1)
        -> Filter: ((st.CrimeRate < (select #2)) and (st.`Code` is not null))  (cost=1.92 rows=50) (actual time=0.094..0.104 rows=23 loops=1)
            -> Table scan on st  (cost=1.92 rows=50) (actual time=0.054..0.060 rows=50 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(State.CrimeRate)  (cost=10.25 rows=50) (actual time=0.024..0.024 rows=1 loops=1)
                    -> Table scan on State  (cost=5.25 rows=50) (actual time=0.010..0.017 rows=50 loops=1)
    -> Filter: (sc.State = st.`Code`)  (cost=14.96 rows=114) (actual time=0.018..0.052 rows=98 loops=23)
        -> Index lookup on sc using schoolstate_idx (State=st.`Code`)  (cost=14.96 rows=114) (actual time=0.018..0.036 rows=98 loops=23)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

Design 2:
CREATE INDEX schoolstate_idx on School(State);
CREATE INDEX state_idx on State(Code);

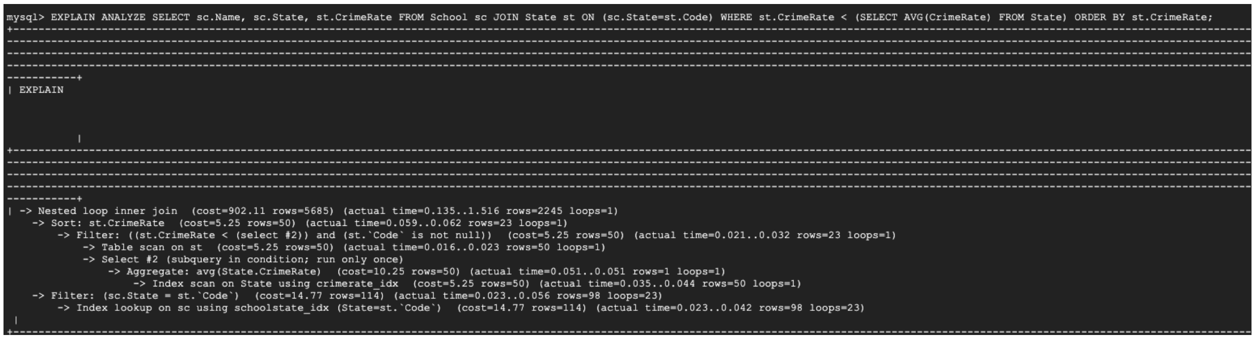
After implementing these indices, the cost is **808.33** and time is similar at 1.49. This is the same compared to design 1. We believe this is because the State table is already very small (51 rows of data) compared to other tables (2000-6000 rows of data). So, creating an index in the State table did not shorten querying as much and had a smaller impact.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;'
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=808.33 rows=5685) (actual time=0.161..1.497 rows=2245 loops=1)
    -> Sort: st.CrimeRate  (cost=1.92 rows=50) (actual time=0.119..0.122 rows=23 loops=1)
        -> Filter: ((st.CrimeRate < (select #2)) and (st.`Code` is not null))  (cost=1.92 rows=50) (actual time=0.081..0.096 rows=23 loops=1)
            -> Table scan on st  (cost=1.92 rows=50) (actual time=0.044..0.052 rows=50 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(State.CrimeRate)  (cost=10.25 rows=50) (actual time=0.022..0.022 rows=1 loops=1)
                    -> Table scan on State  (cost=5.25 rows=50) (actual time=0.009..0.016 rows=50 loops=1)
    -> Filter: (sc.State = st.`Code`)  (cost=14.96 rows=114) (actual time=0.019..0.054 rows=98 loops=23)
        -> Index lookup on sc using schoolstate_idx (State=st.`Code`)  (cost=14.96 rows=114) (actual time=0.018..0.037 rows=98 loops=23)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

Design 3:
CREATE INDEX crimerate_idx on State(CrimeRate);

After indexing, the cost is **902.11** for our query. This is worse. In fact, adding too many indexes can create overhead and slow down our performance. It seems to be more efficient to do a full table scan than create an index for CrimeRate, especially because we're only using CrimeRate to calculate the Avg(CrimeRate) and nothing else. It is not a very important/efficient index to keep.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------+
| EXPLAIN



          |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------+
| -> Nested loop inner join  (cost=902.11 rows=5685) (actual time=0.135..1.516 rows=2245 loops=1)
    -> Sort: st.CrimeRate  (cost=5.25 rows=50) (actual time=0.059..0.062 rows=23 loops=1)
        -> Filter: ((st.CrimeRate < (select #2)) and (st.`Code` is not null))  (cost=5.25 rows=50) (actual time=0.021..0.032 rows=23 loops=1)
            -> Table scan on st  (cost=5.25 rows=50) (actual time=0.016..0.023 rows=50 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(State.CrimeRate)  (cost=10.25 rows=50) (actual time=0.051..0.051 rows=1 loops=1)
                    -> Index scan on State using crimerate_idx  (cost=5.25 rows=50) (actual time=0.035..0.044 rows=50 loops=1)
    -> Filter: (sc.State = st.`Code`)  (cost=14.77 rows=114) (actual time=0.023..0.056 rows=98 loops=23)
        -> Index lookup on sc using schoolstate_idx (State=st.`Code`)  (cost=14.77 rows=114) (actual time=0.023..0.042 rows=98 loops=23)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

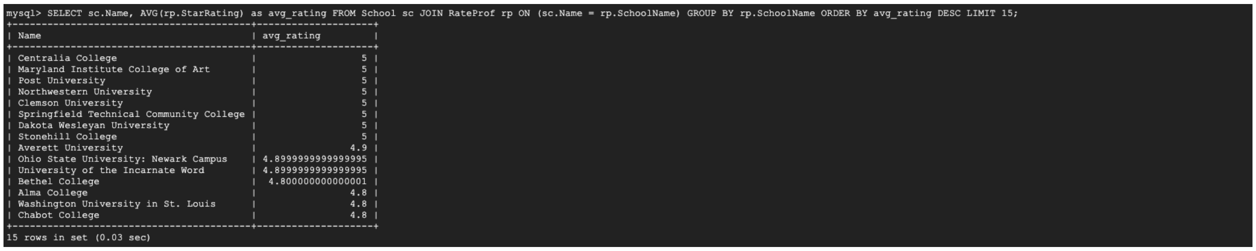
In conclusion, we choose **Design 1** with the lowest cost and fastest lookup.

Advanced SQL Query 2

This is a SQL query that finds the top schools with the highest average RateMyProfessor rating.

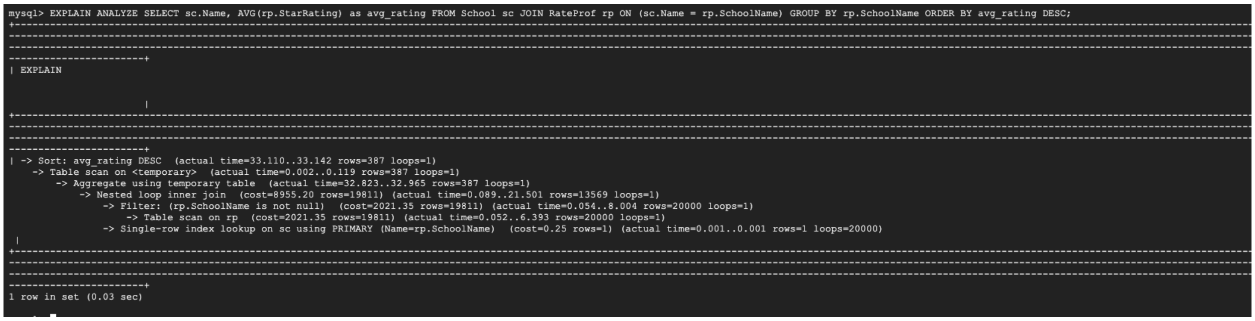
SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;

- Group By
- Join



```
mysql> SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC LIMIT 15;
+-----------------------------------------+--------------------+
| Name                                    | avg_rating         |
+-----------------------------------------+--------------------+
| Centralia College                       |                  5 |
| Maryland Institute College of Art       |                  5 |
| Post University                         |                  5 |
| Northwestern University                 |                  5 |
| Clemson University                      |                  5 |
| Springfield Technical Community College |                  5 |
| Dakota Wesleyan University              |                  5 |
| Stonehill College                       |                  5 |
| Averett University                      |                4.9 |
| Ohio State University: Newark Campus    | 4.8999999999999995 |
| University of the Incarnate Word        | 4.8999999999999995 |
| Bethel College                          |  4.800000000000001 |
| Alma College                            |                4.8 |
| Washington University in St. Louis      |                4.8 |
| Chabot College                          |                4.8 |
+-----------------------------------------+--------------------+
15 rows in set (0.03 sec)
```

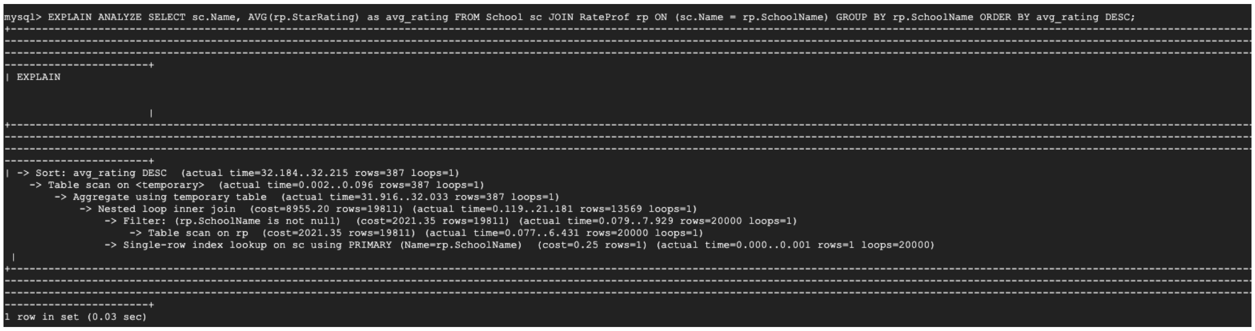
Before indexing, the cost is **8955.20**.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| EXPLAIN


                       |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| -> Sort: avg_rating DESC  (actual time=33.110..33.142 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.119 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=32.823..32.965 rows=387 loops=1)
            -> Nested loop inner join  (cost=8955.20 rows=19811) (actual time=0.089..21.501 rows=13569 loops=1)
                -> Filter: (rp.SchoolName is not null)  (cost=2021.35 rows=19811) (actual time=0.054..8.004 rows=20000 loops=1)
                    -> Table scan on rp  (cost=2021.35 rows=19811) (actual time=0.052..6.393 rows=20000 loops=1)
                -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20000)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
1 row in set (0.03 sec)
```

CREATE INDEX rating_idx ON RateProf(StarRating);

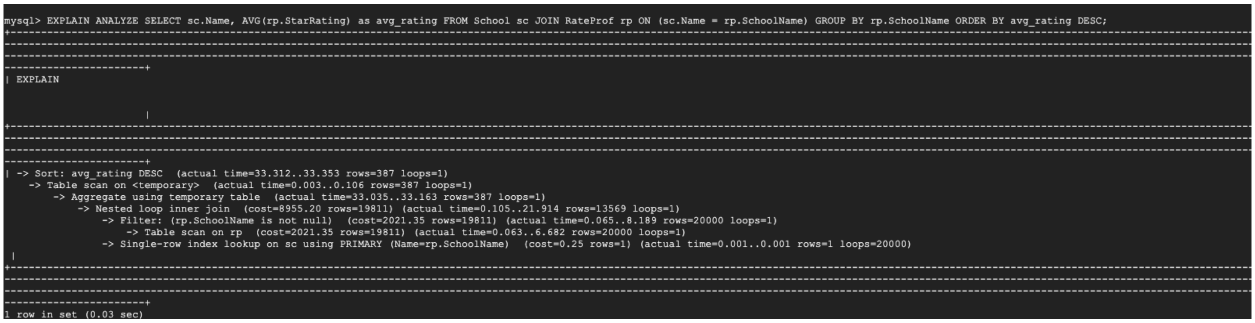
The cost stayed the same at **8955.20**. This is not a very significant difference and the index may be unnecessary.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------+
| EXPLAIN


                      |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------+
| -> Sort: avg_rating DESC  (actual time=32.184..32.215 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.096 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=31.916..32.033 rows=387 loops=1)
            -> Nested loop inner join  (cost=8955.20 rows=19811) (actual time=0.119..21.181 rows=13569 loops=1)
                -> Filter: (rp.SchoolName is not null)  (cost=2021.35 rows=19811) (actual time=0.079..7.929 rows=20000 loops=1)
                    -> Table scan on rp  (cost=2021.35 rows=19811) (actual time=0.077..6.431 rows=20000 loops=1)
                -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName)  (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=20000)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------+
1 row in set (0.03 sec)
```

CREATE INDEX rating_idx ON RateProf(StarRating);
CREATE INDEX name_idx on School(Name);

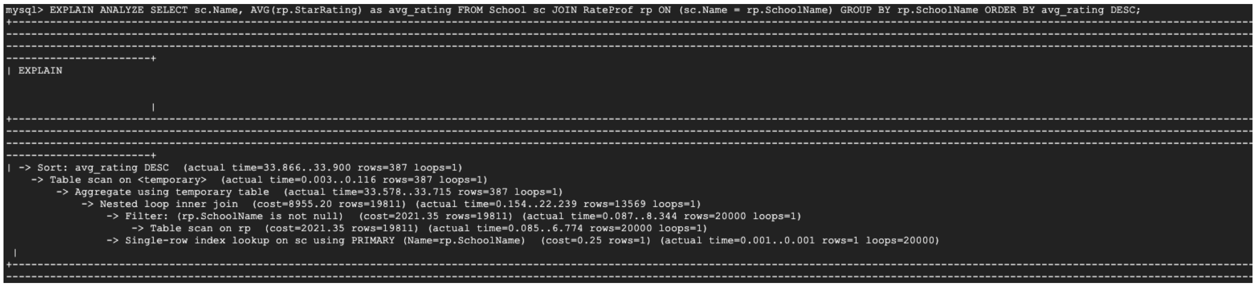
The cost is now **8955.20** which shows that the cost remains the same. So the index appears to make no difference.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------+
| EXPLAIN


                      |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------+
| -> Sort: avg_rating DESC  (actual time=33.312..33.353 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.003..0.106 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=33.035..33.163 rows=387 loops=1)
            -> Nested loop inner join  (cost=8955.20 rows=19811) (actual time=0.105..21.914 rows=13569 loops=1)
                -> Filter: (rp.SchoolName is not null)  (cost=2021.35 rows=19811) (actual time=0.065..8.189 rows=20000 loops=1)
                    -> Table scan on rp  (cost=2021.35 rows=19811) (actual time=0.063..6.682 rows=20000 loops=1)
                -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20000)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------+
1 row in set (0.03 sec)
```

CREATE INDEX rating_idx ON RateProf(StarRating);
CREATE INDEX name_idx on School(Name);
CREATE INDEX rpschool_idx on RateProf(SchoolName);

The cost is now **8955.20** which shows that the cost remains the same. So the index appears to make no difference.



```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| EXPLAIN


                       |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| -> Sort: avg_rating DESC  (actual time=33.866..33.900 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.003..0.116 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=33.578..33.715 rows=387 loops=1)
            -> Nested loop inner join  (cost=8955.20 rows=19811) (actual time=0.154..22.239 rows=13569 loops=1)
                -> Filter: (rp.SchoolName is not null)  (cost=2021.35 rows=19811) (actual time=0.087..8.344 rows=20000 loops=1)
                    -> Table scan on rp  (cost=2021.35 rows=19811) (actual time=0.085..6.774 rows=20000 loops=1)
                -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20000)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

We conclude that no indexes are necessary. The default index is the simplest and efficient design.